VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky

Absolvování individuální odborné praxe
Individual Professional Practice in the Company

2018 **Filip Kössler**

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student:	**Filip Kössler**
Studijní program:	B2647 Informační a komunikační technologie
Studijní obor:	2612R025 Informatika a výpočetní technika
Téma:	Absolvování individuální odborné praxe Individual Professional Practice in the Company
Jazyk vypracování:	čeština

Zásady pro vypracování:

1. Student vykoná individuální praxi ve firmě: Webvalley s.r.o.
2. Struktura závěrečné zprávy:
   a) Popis odborného zaměření firmy, u které student vykonal odbornou praxi a popis pracovního zařazení studenta.
   b) Seznam úkolů zadaných studentovi v průběhu odborné praxe s vyjádřením jejich časové náročnosti.
   c) Zvolený postup řešení zadaných úkolů.
   d) Teoretické a praktické znalosti a dovednosti získané v průběhu studia uplatněné studentem v průběhu odborné praxe.
   e) Znalosti či dovednosti scházející studentovi v průběhu odborné praxe.
   f) Dosažené výsledky v průběhu odborné praxe a její celkové zhodnocení.

Seznam doporučené odborné literatury:

Podle pokynů konzultanta, který vede odbornou praxi studenta.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **doc. RNDr. Petr Šaloun, Ph.D.**

Konzultant bakalářské práce: Bc. Lukáš Vlček

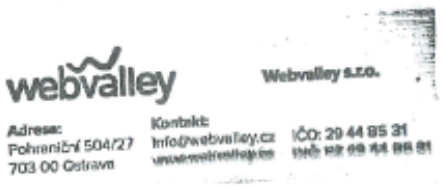
Datum zadání: 01.09.2017
Datum odevzdání: 30.04.2018

doc. Ing. Jan Platoš, Ph.D.	prof. Ing. Pavel Brandštetter, CSc.
vedoucí katedry	*děkan fakulty*

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.

V Ostravě 20. dubna 2018

.....



Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 23. dubna 2018

.....

Tímto bych rád poděkoval všem pracovníků společnosti Webvalley s.r.o. za vytvoření příjemného pracovního prostředí a odbornou pomoc. Především pak Marku Koláči, za cenné rady, jeho železnou trpělivost a vstřícnost při konzultacích.

Abstrakt

V průběhu mé bakalářské praxe ve firmě Webvalley s.r.o. mě bylo přiděleno množství dílčích úkolů na více projektech. Tento dokument obsahuje vylíčení těchto úkolů a postup při jejich řešení. Hlavním úkolem bylo vytvořit řešení, které by zjednodušilo správu a implementaci nástrojů použitých pro přehrávání videí na výše zmíněných projektech a dále pak implementace tohoto řešení.

Klíčová slova: odborná praxe, PHP, Nette, přehrávače videí

Abstract

During my bachelor's degree in Webvalley s.r.o. I have been assigned several partial tasks on multiple projects. This document describes these tasks and how I dealt with them. The main task was to create a solution that would simplify the management and implementation of the tools used to play videos on the above-mentioned projects and then implement the solution.

Key Words: professional practice, PHP, Nette, video players

Obsah

Obsah

Seznam použitých zkratech a symbolů

PHP
- Hypertext Preprocessor

URL
- Uniform Resource Locator je řetězec znaků s definovanou strukturou, který slouží k přesné specifikaci umístění zdrojů informací na Internetu.

HTML
- Hypertext Markup Language

CSS
- Cascading Style Sheets

Seznam výpisů zdrojového kódu

1 Úvod

V tomto dokumentu popisuji průběh své odborné praxe ve firmě Webvalley s.r.o. Pro absolvování odborné praxe jsem se rozhodl, jelikož mnoho zaměstnavatelů klade velký důraz na praktické dovednosti. Odborná praxe je možnost, jak zúročit a prověřit předešlé nabyté teoretické dovednosti. Setkat se s reálnými problémy a jejich řešeními. Pracovat v týmu. Pracovat na úkolech pod tlakem termínů. Seznámit se s fungováním a postupy firmy.

V následujících stránkách budete tedy provedeni obsahem mé praxe, jednotlivých úkolech a jejich řešeními, na kterých jsem pracoval.

2 Popis odborného zaměření firmy, u které student vykonal odbornou praxi a popis pracovního zařazení studenta

2.1 Popis odborného zaměření firmy

Společnost Webvalley s.r.o. byla založena 26.6.2012 a sídlí v Ostravě-Vítkovicích. Převážně cílí na podnikatele z USA, pro které poskytuje softwarová řešení v oblasti webového vývoje, vývoje mobilních aplikací, uživatelského testování, A/B testování a analytik.

2.2 Pracovní zařazení studenta

Ve firmě jsem pracoval jako HTML, CSS kodér a PHP programátor. V průběhu praxe jsem byl nejdříve pověřen úkoly na klientských projektech.

3. Seznam úkolů zadaných studentovi v průběhu odborné praxe

3.1 Seznámení se s PHP frameworkem Nette

Jelikož jsme v byl úplným nováčkem v oblasti PHP frameworků, a firma využívá pro vývoj mnoho svých projektů PHP framework Nette, prvním mým úkolem bylo, se s tímto frameworkem, a jeho architekturou, seznámit.

3.2 Práce na projektech, BrowserStack

Cíl tohoto zadání bylo odvést lehké až středně těžké úkoly na různých projektech, kterou si vyžádal klient. Dalším cílem bylo tyto úpravy otestovat pomocí webového nástroje BrowserStack.

3.3 Seznámení se nástroji pro zobrazení videí Wistia, JW Player a Youtube

Firma na svých projektech hojně využívá multimediální obsah převážně videa. Proto jsem byl pověřen, abych důkladně pročetl dokumentaci a seznámil se s nástroji, které firma pro zobrazení videí používá. Dále jsem byl pověřen, abych se podíval na současné řešení, které firma používá.

3.4 Seznámení se s Google Analytics

Tímto úkolem bylo porozumět, jak funguje Google Analytics, které firma používá ke zlepšení zkušenosti zákazníka.

3.5 Vytvoření nástroje pro práce s videi

V rámci tohoto úkolu bylo vytvořit nové řešení, které by zjednodušilo programátorovi práci s nástroji pro zobrazení videí. Požadavky byly následující:

- lehká přenositelnost mezi projekty a samotná implementace
- Google Anylytics

3.6 Implementace nástroje pro práci s videi na jednotlivých projektech

Po úspěšném splnění předešlého úkolu, jsem byl pověřen implementací nového řešení na projektech.

4. Seznam úkolů zadaných studentovi v průběhu odborné praxe

4.1 Seznámení se s PHP frameworkem Nette

Nette je open source PHP framework pro lehčí a pohodlnější tvorbu webových aplikací. Nette se převážně zaměřuje na eliminaci bezpečnostních rizik a znuvupoužitelnost kódu. Původním autorem frameworku je David Grudl a o jeho další rozvoj se stará organizace Nette Foundation.

4.2 Práce na projektech, testování pomocí BrowserStack

Náplň úkolů bylo ve směs úprava textů, či stylů jednotlivých stránek. Dále optimalizace a opravy na starších prohlížečích, kde jsem se naučil používat speciální zápisy pro Internet Explorer.

BrowserStack je nástroj pro testování webových stránek. Nabízí velké množství reálných zařízení a nativní zkušenost. Má ve svém katalogu přes 1100 desktopových prohlížečů. Dále umožňuje testování přímo z vývojového prostředí.

Účel tohoto úkolu bylo, abych se lépe seznámit se strukturou projektů.

4.3 Seznámení se nástroji pro zobrazení videí Wistia, JW Player a Youtube

Všechny tyto technologie slouží k vložení videa na webové stránky. Wistia jako jediná z těchto technologií také nabízí uchovávání videí na jejich serverech. V rámci tohoto úkolu jsem se seznámil s každým z těchto přehrávačů. Dozvěděl jsem se, jakým způsobem se každý z přehrávačů musí implementovat, jak s nimi pracovat a jaké funkce obsahují.

4.4 Seznámení se s Google Analytics

Google Analytics je služba nabízená společností Google pro analýzu provozu na webových stránkách. Služba byla spuštěna v roce 2005 a momentálně je nejvíce využívaná služba tohoto typu na internetu. Díky této službě je možné sledovat návštěvnost a chování uživatelů.

4.5 Vytvoření nástroje pro práce s videi

Díky předešlým nabytým vědomostem, a zadaných úkolech, jsem věděl, co firma od tohoto nástroje očekává, proto mi byla dána volná ruka při jeho tvorbě, bylo tudíž důležité naplánování celého nástroje. Nástroj má tři vrstvy. První z nich je přehrávač, přesněji jeho parametry a nastavení, druhou vrstvou je Nette komponenta a poslední jsou latte šablony.

Prvním krokem tedy bylo vytvořit abstraktní třídu, pro parametry přehrávačů videí. Tuto třídu jsem nazval *VideoPlayer*. Třída obsahuje proměnné, které jsou společné pro všechny přehrávače (zdroj videa, thumbnail, atd.). Třída taktéž obsahuje proměnnou určující, které šablony budou použity při renderování, tato proměnná je definována v konstruktoru potomka.

V projektech je praxí, že po určité době, která se liší pro každé video, je odkryt skrytý obsah. V šablonách je tato funkcionalita řešena pomocí JavaScript funkce *setTimeout*, jejíž parametr pro určení časového limitu pro spuštění funkce vyžaduje čas v milisekundách, což není v mnoho případech nejpohodlnější řešení pro programátora. Rozhodl jsem se proto programátorům umožnit vkládat čas pomocí srozumitelnějších formátů:

- HH:MM:SS
- MM:SS
- SS

U všech těchto formátu není zapotřebí v jednotlivých částech uvést první číslici, což znamená, že například místo hodnoty *02:09,* může být zadaná hodnota *2:9.* Bylo tedy zapotřebí přidat funkce na validování a konvertovaní vložených hodnot.

Funkce pro validaci času s názvem *validateTime*, nejdříve zkontrolovala, zda vložená hodnota času předávaná parametrem *time,* je typu *string*. Poté jsem ověřil, zda se jedná o správný formát, na což jsem využil regulární výraz:

(^(?:(?:([01]?\d|2[0-3]):)?([0-5]?\d):)?([0-5]?\d)$)

```
1. /**
2.  * @throws Exception time not valid
3.  * @param string $time
4.  */
5. protected static function validateTime($time)
6. {
7.     $pattern = '(^(?:(?:([01]?\d|2[0-3]):)?([0-5]?\d):)?([0-5]?\d)$)';
8.     if (!is_string($time)) {
9.         throw new Exception("TIME \"$time\" has wrong type \"" .
10.            gettype($time) . "\"");
11.    }
12.    if (!preg_match($pattern, $time)) {
13.        throw new Exception("TIME \"$time\" IS NOT VALID IN \"" .
14.            get_called_class() . "\"\n" .
15.            "FORMAT: \"WZ:YX:YX\",\"X:X:X\",\"YX:YX\",\"X:X\", \"YX\", \"X\"\
n" .
16.            "\t X -> [0-9], Y -> [0-5], W -> [0-2], Z -> [0-3]"
17.        );
18.    }
19.}
```

Ukázka 1: Validace času

Za předpokladu, že by vložený čas byl v nesprávném formátu, vyvolal jsem výjimku, která obsahovala nejen nesprávný čas, ale také třídu, v které výjimka vznikla. K tomu jsem použil PHP funkci *get_called_class*, která vrací název třídy, která tuto statickou funkci volá. Programátor bude tedy při použití většího množství přehrávačů na jedné stránce bez problému rozpoznat, kde nastal problém. Tímto způsobem jsem řešil obsah všech výjimek.

Další funkcí byla *convertTimeToMS,* která jak již název napovídá převádí vložený čas na milisekundy, které se používají ve výše zmíněné JavaScriptové funkci. Prvním krokem bylo použití výše zmíněné funkce pro validaci času. Poté jsem z řetězce parsoval hodiny, minuty a vteřiny do jednotlivých proměnných. Poté jsem zjistil, o který formát času se jedná a podle toho upravil výpočet na vteřiny.

```
1. /**
2.  * Converts time to ms.
3.  *
4.  * FORMAT:
5.  *      "HH:MM:SS"
6.  *      "MM:SS"
7.  * @param string $time
8.  * @return integer
9.  * @throws Exception
10. */
11.protected static function convertTimeToMS($time)
12.{
13.    self::validateTime($time);
14.
15.    $hours = $minutes = $seconds = null;
16.    sscanf($time, "%d:%d:%d", $hours, $minutes, $seconds);
17.
18.    if (isset($seconds)) {
19.        return ($hours * 3600 + $minutes * 60 + $seconds) * 1000;
20.    } else if (isset($minutes)) {
21.        return ($hours * 60 + $minutes) * 1000;
22.    } else {
23.        return $hours * 1000;
24.    }
25.}
```

Ukázka 2: Konvertování času na milisekundy

Pro převedení času z milisekund jsem použil PHP funkci *gmdate,* s formátem *H:i:s.*

```php
/**
 * Convert ms to time.
 * @param integer $ms
 * @return string
 */
protected static function convertMSToTime($ms)
{
    if ($ms === null) {
        return null;
    } else {
        return gmdate("H:i:s", $ms / 1000);
    }
}
```

Ukázka 2: Konvertování času z milisekund

Kromě těchto tří funkcí třída také obsahovala 2 abstraktní. Funkce *validateProperties,* v které se provádí podle specifických potřeb každého přehrávače validace vložených dat a funkce *getProperties*, která vrací všechny parametry zabalené v poli.

Po vytvoření této abstraktní třídy, jsem vytvořil třídy pro jednotlivé přehrávače Wistia, JW Player a Youtube. Vytvoření a nastavení parametrů přehrávače vypadá takto:

```php
$ytPlayer = new YTPlayer();
$ytPlayer->setThumbnail(false)
    ->setControls(false)
    ->setAutoPlay(false)
    ->setIsRapid(true)
    ->setRapidTimer('30')
    ->setAnalytics(VideoPlayer::ANALYTICS_BASIC)
    ->setAnalyticsIdentifier('avoid')
    ->setIsFbTimer(true)
    ->setFbThreshold("5:00");
```

Ukázka 3: Konvertování času na milisekundy

Dalším krokem, po vytvoření tříd pro přehrávače bylo vytvoření Nette komponenty *VideoControl*. Jelikož nejlepším způsobem použití komponent je Dependenci Injection, nejdříve jsem vytvořil interface *IVideoControl* a zaregistroval komponentu jako službu do konfiguračního souboru frameworku.

Na projektech se mnohdy na jedné stránce používají jiné přehrávače pro různé platformy. Na jedné stránce může být více videí. Implementace všech videí na jedné stránce musí být na sobě nezávislá. Všechny tyto proměnné a další jsem musel vzít v potaz při návrhu komponenty. Každá vytvořená komponenta tudíž obsahuje identifikátor, který je předáván do šablon. Dále kvůli

identifikaci platforem je v konstruktoru komponenty volána třída *Browser*, která analyzuje klientské zařízení a je uložena do statické proměnné třídy.

```
1. /**
2.  * VideoControl constructor.
3.  * @param string $videoID Sets video ID.
4.  * @throws Exception
5.  */
6. public function __construct($videoID)
7. {
8.     parent::__construct();
9.     if (self::$browser == null) {
10.         self::$browser = new Browser();
11.     }
12.     self::setVideoID($videoID);
13.}
```

Ukázka 4: Konstruktor třídy VideoControl

Rozhraní komponenty tvoří 4 funkce. *SetPlatformPlayer*, *getPlatformPlayer*, *clonePlatform* a *removePlatform*. První funkce slouží pro vložení přehrávače pro určitou platformu. Přípustné platformy jsou definované pomocí veřejných konstant uvnitř třídy, jsou definovány takto: *ALL*, *DESKTOP*, *MOBILE*, *I_PAD*, *I_PHONE*, *ANDROID*. Jak vložená platforma, tak přehrávač jsou validovány. Platforma je validována proti definovaným platformám třídy, přehrávač musí být instancí třídy *VideoPlayer*. Po validaci je přehrávač uložen do pole, v kterém je klíčem platforma a hodnotou přehrávač. Druhá funkce je pro získání přehrávače pro určitou platformu. Třetí funkce z jedné platformy klonuje přehrávač do druhé. Čtvrtá a poslední funkce odstraní přehrávač pro danou platformu.

Komponenta obsahuje 3 render funkce, které se přímo volají v Latte šabloně. *RenderHeader* se používá v hlavičce dokumentu stránky. Používá se převážně k vložení CSS stylů pro videa. *RenderHtml* se volá v těle dokumentu a vkládá video samotné a jeho obal. *RenderJs* se volá na konci těla dokumentu, kde jsou typicky umístěny *JavaScript* kódy, které funkce obsahuje.

Nyní si projdeme proces renderování. První funkcí, která se tedy zavolá je *renderHeader*. V této funkci probíhá množství dalších procesů. Nejdříve se validují vložené platformy. Pro splnění musí být definována minimálně platforma *ALL*, nebo platformy *DESKTOP* a *MOBILE*. Následuje proces určení současné platformy pomocí výše zmíněného nástroje *Browser*. Proces dává prioritu individuálním platformám což znamení, že například při definovaných platformách *ALL* a *I_PAD* a zákazník používá iPad bude vybrána platforma *I_PAD*. Vybraná hodnota se poté uloží do proměnné instance.

```php
/**
 * Determines witch platform shall be used depending on clients platform and
 * defined platforms.
 *      Individual platforms has higher priority (iPhone, ...)
 *      Platform MOBILE and DESKTOP has higher priority then ALL.
 *
 * @throws Exception
 * @return string Returns platform used for current render.
 */
private function determinatePlatform()
{
    self::validatePlatforms();

    $mobile = self::$browser->isMobile() || self::$browser->isTablet();
    if ($mobile) {
        $platform = self::$browser->getPlatform();
        if ($platform == Browser::PLATFORM_IPHONE &&
            $this->setUp['I_PHONE'] != null) {
            return self::PLATFORM_I_PHONE;
        } else if ($platform == Browser::PLATFORM_IPAD &&
            $this->setUp['I_PAD'] != null) {
            return self::PLATFORM_I_PAD;
        } else if ($platform == Browser::PLATFORM_ANDROID &&
            $this->setUp['ANDROID'] != null) {
            return self::PLATFORM_ANDROID;
        } else if ($this->setUp['MOBILE'] != null) {
            return self::PLATFORM_MOBILE;
        } else {
            return self::PLATFORM_ALL;
        }
    } else {
        if ($this->setUp['DESKTOP'] != null) {
            return self::PLATFORM_DESKTOP;
        } else {
            return self::PLATFORM_ALL;
        }
    }
}
```

Ukázka 5: Určení výsledné platformy

Po získané platformě je dále uložen do instanční proměnné také samotný přehrávač.

Jelikož by bylo nechtěné některé věci z šablon jednotlivých přehrávačů renderovat vícekrát (např. CSS styly), další část kódu má za úkol vyřešit tuto problematiku. Třída *VideoControl* má třídní proměnnou *includedPlayers* a instanční proměnnou *includeOnce*. Do proměnné *includedPlayer* se ukládají třídy přehrávačů, které již byly vyrenderovaní, za předpokladu, že žádný přehrávač neprošel tímto procesem, nebo není přítomen v poli, nastaví se proměnná *includeOnce* boolean hodnotou *true*, jinak *false*. To umožňuje v šablonách pro jednotlivé přehrávače určit, zda se má prvek renderovat, či ne. Další funkcí, která je nastavení vývojářského modu, za předpokladu, že v *URL* je přítomen parametr *dev-video* s hodnout *true*, bude tento mód aktivován. V praxi to znamená, že *JavaScript* vypíše do konzole veškeré parametry a nastavení přehrávače.

Po veškeré této přípravě přichází krok vkládání proměnných do šablony. Nejdříve se vybere však šablona samotná. K výběru se použije proměnná samotného přehrávače, o které jsem se zmiňoval na

začátku kapitoly. Poté se do šablony vloží všechny potřebné parametry pomocí funkce přehrávače *getProperties*.

```
1. /**
2.  * Renders Header part of video player.
3.  */
4. public function renderHeader()
5. {
6.     self::determinatePlayer();
7.     $this->getParams();
8.
9.     $template = $this->template;
10.    $template->setFile(__DIR__ . "/templates/" .
11.        $this->player->getTemplateFolder() . "/header.latte");
12.
13.    $template->videoID = $this->videoID;
14.    $template->includeOnce = $this->includeOnce;
15.
16.    foreach ($this->player->getProperties($this-
>platform) as $key => $value) {
17.        if ($value !== null) {
18.            $template->$key = $value;
19.        }
20.    }
21.
22.    $template->platform = $this::$browser->getPlatform();
23.    $template->render();
24.}
```

Ukázka 6: Implementace funkce renderHeader

Další render funkcí komponenty je *renderHtml*. Tato funkce na rozdíl od předchozí jen vkládá parametry do šablony.

Poslední funkcí je *renderJS*. Tato funkce opět vkládá parametry do šablon. Avšak je zde předávána jedna proměnná navíc. Za předpokladu, že je aktivován developer mód. Vloží se též do šablony pole se všemi parametry, které pak *JavaScript* vypíše do konzole.

Nyní se podíváme na poslední vrstvu celého nástroje – šablony. Jelikož každý přehrávač v zásadě potřebuje jinou implementaci šablon, každý přehrávač má svoje vlastní šablony, které odpovídají počtu renderovacích funkcí třídy *VideoControl*. Každý přehrávač má tedy šablonu pro hlavičku, tělo a Javascript kód. V šabloně hlavičky se většinou vykreslují jenom CSS styly přehrávače.

```
1. <link n:if="$includeOnce" rel="stylesheet" href="{$basePath}/www/css/controls
/video/yt.css?v=1.1">
```

Ukázka 7: Šablona hlavičky

V kódu je vidět, že je použita proměnná *includeOnce* jelikož není potřeba mít stejné styly na stránce vícekrát.

Každé video je obalenou je obaleno HTML strukturou, která zaručuje, že video bude mít poměr stran 16:9. Dále se v šabloně vkládá unikátní identifikátor, který se definoval v konstruktoru třídy *VideoControl*. V šabloně se též podle použitého přehrávače videa nastavuje zdroj videa, ovládací prvky přehrávače, thumbnail a další.

```
1. <div style="padding:56.25% 0 0 0;position:relative;">
2.     <div style="height:100%;left:0;position:absolute;top:0;width:100%;">
3.         <iframe id="{$videoID|noescape}"
4.                 style="height:100%;width:100%"
5.                 src="https://www.youtube.com/embed/{$source|noescape}?rel=0&c
ontrols={$controls ? '1' : '0'}&showinfo=0&enablejsapi=1"
6.                 frameborder="0" allowfullscreen>
7.         </iframe>
8.
9.         {if $thumbnail || !$controls}
10.        <div id="play-button-{$videoID|noescape}"
11.             class="yt-thumbnail"
12.             style="{if $thumbnail}background: url({$thumbnail|noescape}); ba
ckground-size: cover;{/if}">
13.        </div>
14.        {/if}
15.        <div class="youtube-overlay"></div>
16.    </div>
17.</div>
```

Ukázka 8: Šablona těla

V kódu výše je šablona přehrávače Youtube. Youtube používá pro vkládání videí HTML element iframe. Jako id elementu je nastaven unikátní identifikátor třídy *VideoControl*. Dále ve zdroji elementu, je vidět použití zdroje a nastavení ovládacích prvků přehrávače. Jelikož Youtube nemá zabudovaný thumbnail, video překrývá element, který má jako své pozadí chtěný thumbnail a také slouží jako první tlačítko pro přehrávání videa.

U předchozí šablony jsem již využil znalosti nabyté studiem jednotlivých přehrávačů, avšak u poslední šablony zabývající se *JavaScriptem* jednotlivých přehrávačů, jejich API, jsem je využil na plno. Jako příklad si opět vezměme Youtube player. Bylo zapotřebí správně nakonfigurovat Youtube JavaScript API a události na různé akce videa. Každá událost musela obsahovat unikátní identifikátor získaný z proměnných šablony, aby bylo více videí v dokumentu na sobě nezávislých. Tyto události byli důležité převážně kvůli Google Analytics, ale také dalším funkcím.

Nejdříve jsem tady vytvořil funkci pro inicializaci Youtube videa tato funkce se na stránku vykreslovala jenom jednou. Funkce definuje událost, kdy je přehrávač pozastaven, kdy je spuštěn a v neposlední řadě, kdy je video připraveno. U této události se také vyvolá funkce, která posílá každou vteřinu událost o pozici přehrávače, tato informace se pak používá v Google Analytics.

```
1. <script n:if="$includeOnce" type="text/javascript">
2.     function youtubeSetUp(videoID, source, autoPlay) {
3.         var player = new YT.Player(videoID, {
4.             width: '100%',
5.             videoId: source,
6.             events: {
7.                 'onReady': function (event) {
8.                     $(document).trigger('player.' + videoID + '.loaded');
9.                 },
10.                'onStateChange': function (event) {
11.                    if (event.data === YT.PlayerState.PLAYING) {
12.                        $(document).trigger('player.' + videoID + '.started')
;
13.                    }
14.                    if (event.data === YT.PlayerState.PAUSED) {
15.                        $(document).trigger('player.' + videoID + '.paused');

16.                    }
```

```
                }
            }
        });

        $(document).on('player.{$videoID|noescape}.loaded', function () {
            // every second fire the playing event
            setInterval(function () {
                // broadcast the event
                $(document).trigger('player.' + videoID + '.playing', [{
                    playbackTime: player.getCurrentTime(),
                    videoLength: player.getDuration()
                }]);
            }, 1000);
        });
    }
</script>
```

Ukázka 9: Funkce pro inicializaci Youtube videa

Jelikož bylo zapotřebí více přehrávačů v jednom dokumentu. Musel jsem vytvořit kód, který vytvoří pole přehrávačů, které při načtení Youtube API tyto videa inicializuje. Při implementaci jsem použil proměnnou šablony *includeOnce*, ta mi umožnila provést inicializaci API jenom jednou, poté použít předdefinovanou funkci z API *onYoutubeIframeAPIReady*, kterou volá samo API za předpokladu, že se API načte. Mezitím, ze se API načítá, vkládám do pole jednotlivé konfigurace videí. V této šabloně se také vkládá šablona obsahující Google Anylytics a další funkce.

```
<script type="text/javascript">
    {if $includeOnce}
    var tag = document.createElement('script');
    tag.id = 'youtube-script';
    tag.src = 'https://www.youtube.com/iframe_api';
    var firstScriptTag = document.getElementsByTagName('script')[0];
    firstScriptTag.parentNode.insertBefore(tag, firstScriptTag);

    var ytSetUps = [];
    ytSetUps.push(['{$videoID|noescape}', '{$source|noescape}', {if $autoPlay} true {else} false {/if}]);

    function onYouTubeIframeAPIReady() {
        for (var ytSetUp in ytSetUps) {
            youtubeSetUp(ytSetUps[ytSetUp][0], ytSetUps[ytSetUp][1], ytSetUps[ytSetUp][2]);
        }
    }
    {else}
    ytSetUps.push(['{$videoID|noescape}', '{$source|noescape}', {if $autoPlay} true {else} false {/if}]);
    {/if}
</script>

{include "../tracking.latte"}
```

Ukázka 10: Funkce pro inicializace Youtube JavaScript API

Posledním bodem celého úkolu byla implementace Google Analytics. Každou vteřinu, kdy se pozice přehrávače zvýšila bylo zapotřebí poslat událost. Aby si navzájem videa neovlivňovala počet dosažených procent, napsal jsem anonymní funkci, která se sama vykonává. Při zvýšení dosažených procent, se odesílá událost obsahující informace o identifikátorech videa a dosažených procent.

```
{if $analytics === 'A_BASIC'}
    <script type="text/javascript">
        (function () {
            var sentPercentage = [];

            $(document).on('player.{$videoID|noescape}.playing', function (event, data) {
                var percentage = Math.floor(data.playbackTime / data.videoLength * 100);
                percentage = percentage > 100 ? 100 : percentage;
                var len = 3;
                len -= percentage.toString().length;
                while (len--)
                    percentage = '0' + percentage.toString();

                if (-1 === $.inArray(percentage, sentPercentage)) {
                    if(window.ga)
                        ga('send', 'event', '{$analyticsIdentifier|noescape}-{$videoID|noescape}-action', 'playback', percentage + '%');
                    sentPercentage.push(percentage);
                }
            });
        })();
    </script>
{elseif $analytics === 'A_ADVANCED'}
    <script type="text/javascript">
    </script>
{/if}
```

Ukázka 11: Google Anylytics

Nakonec jsem sepsal dokumentaci k celému řešení popisující, jakým způsobem se komponenta má používat.

4.6 Implementace nástroje pro práci s videi na jednotlivých projektech

Jelikož jako tvůrce nástroje jsem byl nejlepším kandidátem pro jeho nasazení, byl jsem tímto úkolem také pověřen. Prvním krokem bylo přidat komponentu do projektu takového. Dále jsem zaregistroval komponentu jako službu do konfiguračního souboru Nette frameworku daného projektu.

Nette *Presenter*, je komponenta, která reprezentuje instanci webové stránky, pro každou takovou instanci jsem nejdříve vytvořil proměnnou, do které je následně použitím *Dependency Injection*, vložil mnou vytvořenou komponentu.

```
/** @var IVideoControl @inject */
    public $videoControl;
```

Ukázka 12: Použití Dependecy Injection

Dalším krokem, bylo podle frameworkem určeného způsobu vytvořit funkci, která samotnou komponentu vytváří a v našem případě v ní také probíhá nastavení potřebných přehrávačů a jejich následovné vložení do komponenty pod potřebnou platformu.

```
1.  public function createComponentVideoControl()
2.   {
3.       $videoControl = $this->videoControl->create('video');
4.
5.       $ytPlayer = new YTPlayer();
6.       $ytPlayer->setThumbnail(false)
7.           ->setControls(false)
8.           ->setAutoPlay(false)
9.           ->setIsRapid(true)
10.          ->setRapidTimer('30')
11.          ->setAnalytics(VideoPlayer::ANALYTICS_BASIC)
12.          ->setAnalyticsIdentifier('avoid')
13.          ->setIsFbTimer(true)
14.          ->setFbThreshold("5:00");
15.
16.      $videoControl->setPlatformPlayer(VideoControl::PLATFORM_I_PHONE, $ytPlayer);
17.
18.      return $videoControl;
19.  }
```

Ukázka 13: Vytvoření komponenty a její nastavení

Každý prezenter může obsahovat více množství funkcí pro renderování, což umožňuje měnit obsah jedné webové stránky. Firma tuto možnost, co se videí používá například pro změnu zdroje videa podle lokace zákazníka. V praxi se proto při implementaci ve funkci pro vytvoření komponenty nastavili vlastnosti přehrávače, které byli stejné pro všechny render funkce prezenteru a ostatní nastavovali, nebo přepisovali přímo v jednotlivých render funkcích. Render funkce prezenteru obsahuje metodu, díky které je možné získat komponentu a dále s ní pracovat.

```
1.  public function renderDefault()
2.  {
3.          ...
4.          // Getting component and players
5.          /** @var VideoControl $defaultVideo */
6.          $defaultVideo = $this->getComponent('videoControl');
7.          /** @var VideoPlayer $ytDefault */
8.          $ytDefault = $defaultVideo
9.              ->getPlatformPlayer(VideoControl::PLATFORM_I_PHONE);
10.
11.         $ytDefault->setSource('ntbcok1m98');
12.         ...
13. }
```

Ukázka 14: Použití komponenty v render funkci prezenteru

Tímto jsme tady implementoval komponentu, ale stále bylo ještě zapotřebí na správných místech v šabloně webové stránky použím render funkce komponenty – *renderHeader*, *renderHtml*, *renderJs*.

```
</head>
    ...

    {control videoControl:Header}
    ...

</head>
<body>
    ...
    {control videoControl:HTML}
    ...
    {control videoControl:JS}
    ...
</body>
```

Ukázka 14: Použití render funkcí komponenty VideoControl v šabloně

Celý proces implementace je rychlý a jednoduchý, avšak samotné nasazení na reálných projektech již tak přívětivé nebylo. Práci ztěžovala komplexnost a provázanost jednotlivých projektů a jejich prvků, v kterých se používá mnohonásobná dědičnost nejen v PHP kódu, ale také v šablonovacím systému takovém. Jelikož součástí nasazení také bylo odstranit kód, který byl používaný dříve, musel jsem si dávat pozor, aby mé změny neovlivnily desítky dalších stránek v projektu, což v důsledku znamenalo pomalou a pečlivou práci, která vyžadovala hlubokou znalost projektů a neustálé testování.

5 Získané a využité zkušenosti a znalosti

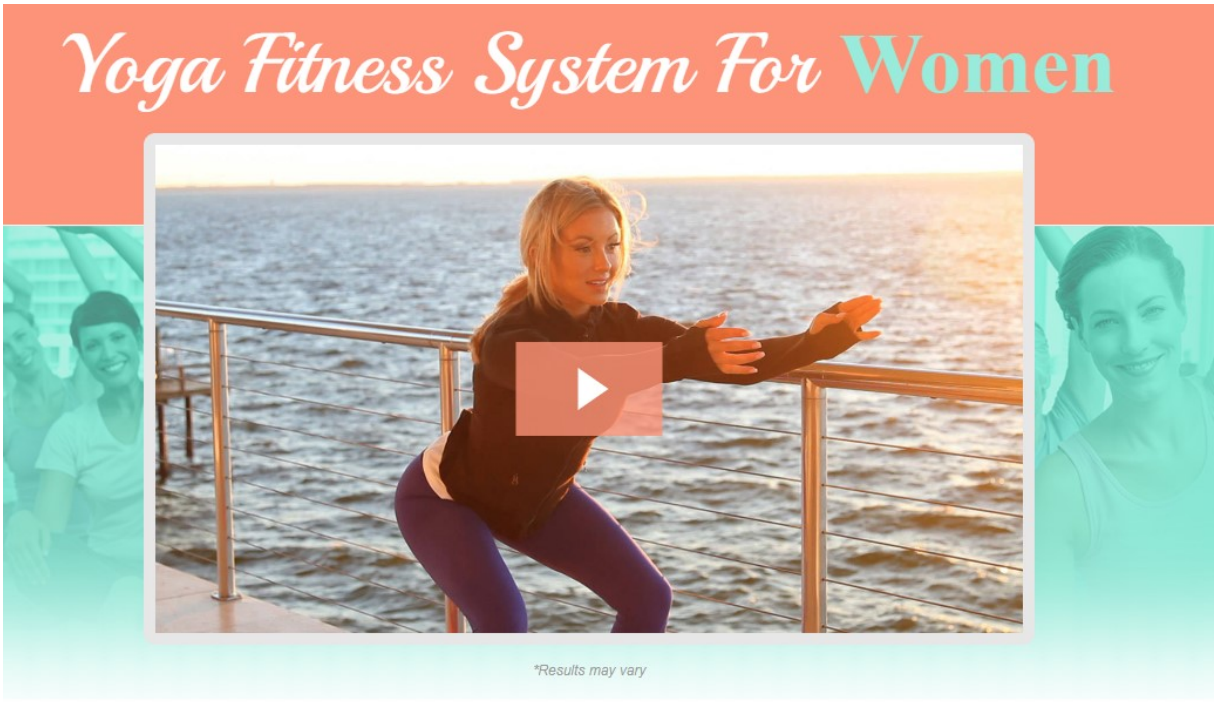
Za dobu vykonávání praxe jsem využil znalosti většina programovacích předmětů, především předmět zabývající se objektově orientovaným programováním. Získal jsem nové zkušenosti s programovacím jazykem PHP. Seznámil jsem se blíže s PHP frameworkem Nette. Naučil jsem se pracovat s nástrojem pro verzování GIT. Osvojil jsem si práci v týmu, při které jsem aktivně komunikoval se členy nejen při řešení problémů. Dále jsem také hojně využil znalosti s programovacím jazykem JavaScript, které jsem získal z předmětu Tvorba aplikací pro mobilní zařízení I, tyto znalosti jsem dále prohloubil. V neposlední řadě jsem prohloubil své dovednosti v oblasti kódování responzivního designu.

6 Dosažené výsledky v průběhu odborné praxe a její celkové zhodnocení

V průběhu praxe jsem značně prohloubil své znalosti v oblasti vývoje webových aplikací, o kterých jsem ze začátku praxe nevěděl skoro nic. Naučil jsem se používat nové technologie jako GIT a Nette Framework. Cením zkušeností získané prací na skutečných problémech a projektech. Naučil jsem se přistupovat ke komplexním problémům a vyřešit je. Všechny zadané úkoly se mi podařilo bez větších problémů splnit s odbornou pomocí a rad mých spolupracovníků. Podařilo se mi vytvořil komplexní nástroj, který programátorům v budoucnu ulehčí práci a zlepší produktivity a přehlednost kódu.

Ukázka webové stránky používající komponentu VideoControl

Desktop (přehrávač Wistia)



iPhone (přehrávač Youtube)